|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Радиотехнический (РТ)

КАФЕДРА Системы обработки информации и управления (РТ-5)

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

**\_\_\_\_\_\_Реализация алгоритма Хаффмана\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ на Turbo Assembler \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_**

Студент РТ5-41Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Забурунов Л. В.

(Группа) (Подпись, дата) (Фамилия И.О.)

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_**Семёнов Д. В.

(Подпись, дата) (Фамилия И.О.)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (Фамилия И.О.)

*2020г.*

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой \_\_\_\_\_\_\_\_\_\_

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(И.О.Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_\_ г.

**ЗАДАНИЕ**

**на выполнение курсового проекта**

по дисциплине Системное программирование

Студент группы РТ5-41Б Забурунов Леонид Вячеславович

(Фамилия, имя, отчество)

Тема курсового проекта Реализация алгоритма Хаффмана на Turbo Assembler

Направленность КП (учебный, исследовательский, практический, производственный, др.)

Учебная

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения проекта: 25% к 3 нед., 50% к 9 нед., 75% к 12 нед., 100% к 15 нед.

**1. Задание**

Создать программу, осуществляющую кодирование и декодирование по кодам Хаффмана. Программа должна быть реализована без помощи языков высокого уровня, разрешается использование только языка ассемблера.

**2. Оформление курсовой работы**

Расчётно-пояснительная записка на \_\_ листах А4.

Дата выдачи задания «12» февраля 2020 г.

**Руководитель курсового проекта**  Семёнов Д. В.

(Подпись, дата) (Фамилия И.О)

**Студент**   Забурунов Л. В.

(Подпись, дата) (Фамилия И.О)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

**[1.](#_Toc42453957)****[Введение](#_Toc42453957)** [4](#_Toc42453957)

[**2.** **Основная часть курсовой работы** 5](#_Toc42453958)

[**2.1 Кодирование** 5](#_Toc42453959)

[2.1.1 Чтение строки 7](#_Toc42453960)

[2.1.2 Подсчёт частоты встречаемости символов 10](#_Toc42453961)

[2.1.3 Сортировка по частоте встречаемости 15](#_Toc42453962)

[2.1.4 Построение дерева 18](#_Toc42453963)

[2.1.5 Получение кодировки 29](#_Toc42453964)

[2.1.6 Кодирование исходной строки 36](#_Toc42453965)

[**2.2** **Декодирование** 41](#_Toc42453966)

[**3.** **Примечания** 47](#_Toc42453967)

[**4.** **Полный листинг программы** 48](#_Toc42453968)

[**4.1** **Файл macro.asm** 48](#_Toc42453969)

[**4.2** **Файл print\_m.asm** 61](#_Toc42453970)

[**4.3** **Файл coding.asm** 66](#_Toc42453971)

[**4.4** **Файл decoding.asm** 68](#_Toc42453972)

# **Введение**

Алгоритм Хаффмана является алгоритмом оптимального кодирования последовательности символов. Опорой данного алгоритма является бинарное кодовое дерево, обход по которому позволит получить код для каждого символа, входящего в исходную последовательность. Как правило, для реализации используются сложные структуры данных (связные списки различных типов, очереди с приоритетами; даже само понятие «структура»). Сложность (и возникающий от неё интерес) данного задания заключается как раз в том, что для реализации на Turbo Assembler (далее - TASM) придётся следить за каждым своим действием, используя лишь простейшие средства машинного языка.

# **Основная часть курсовой работы**

# **2.1 Кодирование**

Основная идея алгоритма Хаффмана – присвоение кодов наименьшей длины тем символам, которые встречаются наиболее часто. Для этого используется следующее:

А) считывание строки и анализ количества появлений в ней каждого символа;

Б) упорядочивание полученных значений;

В) построение дерева по следующему правилу:

В.1) листами считаются ячейки полученных значений (символ и соответствующее значение встречаемости);

В.2) берётся два листа с наименьшим значением и создаётся их общий предок, имеющий значение, равное сумме двух потомков;

В.3) повторение шага в.2 до тех пор, пока не получим корень дерева – вершина со значением, равным длине строки;

В.4) для двух дуг, выходящих из каждого узла, нужно поставить в соответствие числа 0 и 1 (обычно считают, что «левая» дуга – это ноль, «правая» - единица);

В.5) код символа – цифры, встретившиеся на пути из корня дерева до листа, соответствующего данному символу.

Г) перекодирование строки на основе полученного дерева.

Таким образом мы получаем последовательность шагов:

1. Считать строку;
2. Подсчитать количество вхождений каждого символа в считанную строку;
3. Отсортировать по количеству вхождений;
4. Построить кодовое дерево в соответствии с правилами пункта В;
5. Совершить обход дерева и получить коды для каждого символа;
6. Заменить исходную строку на закодированную.

Пункты 1 – 3 и 6 не вызывают вопросов и их реализация будет сравнительно простой. На пунктах 4 и 5 остановимся поподробнее, когда до них дойдёт очередь.

## Чтение строки

Определимся с тем, что исходная строка будет храниться в файле на компьютере (закодированная строка так же будет выводиться в файл).

Напишем небольшую программу на TASM, которая считывает из файла строку и затем выводит её в DOS-консоль. Файл макросов:

openfile MACRO filename

push dx; сохраняем параметр, находившиеся в регистре до вызова функции

mov ax, 3d00h; код функции

mov dx, offset filename; параметры функции

int 21h; вызов прерывания

pop dx; восстанавливаем параметр

ENDM

readfile MACRO handle, count, buffer

push ax;

push bx;

push cx;

push dx;

mov ah, 3fh;

mov bx, handle;

mov cl, count;

mov dx, offset buffer;

int 21h;

pop dx;

pop cx;

pop bx;

pop ax;

ENDM

; Вывести символ

printc MACRO C

push ax;

push dx;

xor ax, ax;

xor dx, dx;

mov ah, 02h;

mov dl, C;

int 21h;

pop dx;

pop ax;

ENDM

; Вывести строку

prints MACRO S

local printString, endOfString;

push ax;

push si;

xor ax, ax;

xor si, si;

local char: byte; локальная переменная в рамках макроса

mov si, 0;

printString:

mov ah, S + si;

cmp ah, 24h; признак конца строки - код символа знака доллара

je endOfString; если конец строки, то цикл из безусловных прыжков прерывается

mov char, ah;

printc char; пользуемся собственным макросом

inc si;

jmp printString;

endOfString:

pop si;

pop ax;

ENDM

Основная программа:

INCLUDE \..\Programs\macro.asm

.MODEL SMALL

.STACK 100h

.DATA

buffer DB 512 dup ('$')

fileName DB 'Huffman.txt', 0

fileHandle DW ?

\_length DB 22

.CODE

START:

; Инициализация сегмента данных и базовая очистка регистров перед началом работы

mov ax, @Data

mov ds, ax

xor ax, ax

xor bx, bx

xor cx, cx

xor dx, dx

xor si, si

xor di, di

openfile fileName

mov fileHandle, ax; после работы макроса в регистре AX находится дескриптор файла

readfile fileHandle, \_length, buffer

prints buffer

; Выход из программы

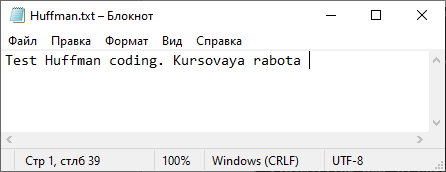
mov ax, 4c00h;

int 21h;

END START;

Результат работы:

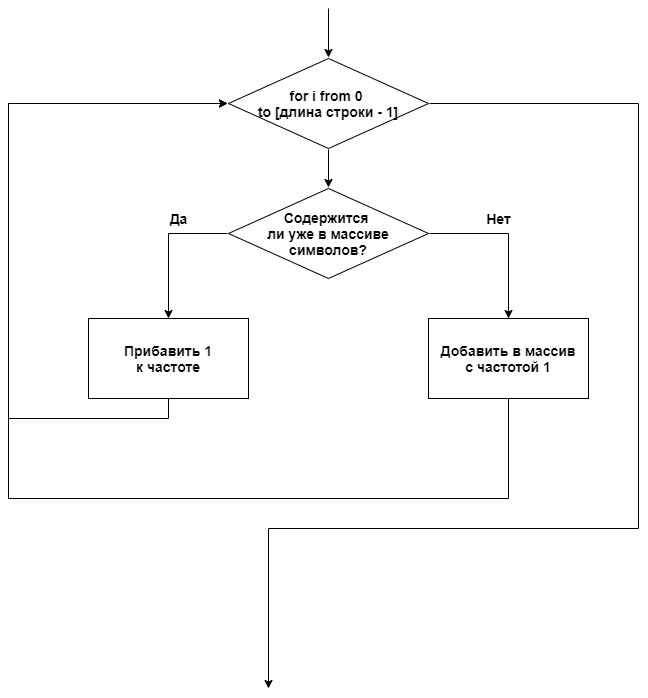




Процедура чтения из файла работает; сохраняем её в свой актив.

## Подсчёт частоты встречаемости символов

Теперь после чтения строки нужно совершить проход по буферу данных и подсчитать количество вхождений каждого символа в строку. Общая блок-схема:



С этого момента приходится учитывать особенности ассемблера. Поскольку создание сложных структур данных затруднительно, мы будем использовать два одномерных массива – для символов и для частоты. Ограничимся для нашего проекта длиной этих массивов в 70 (алфавит английского языка, цифры, знаки препинания и прочее).

Текст созданного макроса:

freq\_co MACRO string, c\_array, f\_array, len

local freqCount, endOfString, found, not\_found, cont, checking, found\_c, nextStep; локальные метки

push ax;

push bx;

push dx;

push di;

push si;

push bp;

local c\_length: byte, char: byte; локальные переменные

mov c\_length, 1;

mov si, 0;

freqCount:

mov ah, string + si;

cmp ah, 0; признак конца строки - код символа знака доллара

je endOfString; если конец строки, то цикл из безусловных прыжков прерывается

; Сверяем со всеми находящимися в массиве символами

mov char, ah;

mov di, 0;

mov dl, c\_length;

checking:

mov ah, c\_array + di;

cmp ah, char;

je found\_c; совершаем прыжок, если нашлось сравнение

inc di;

cmp di, dx;

jne checking;

mov bx, 0;

jmp nextStep;

found\_c:

mov bx, 1;

nextStep:

; На основе значения BX совершаем дальнейшие действия

cmp bx, 1;

je found;

cmp bx, 0;

je not\_found;

found:

inc f\_array + di; увеличиваем частоту символа на 1

jmp cont;

not\_found:

inc c\_length; новый символ

mov di, dx; в DL сидит c\_length до работы инкрементора

mov dl, char;

mov c\_array + di - 1, dl; заносим символ в массив

mov f\_array + di - 1, 1; и частоту

jmp cont;

cont:

inc si;

jmp freqCount;

endOfString:

; Запоминаем длину для будущих операций

mov al, c\_length;

dec al;

mov len, al;

pop bp;

pop si;

pop di;

pop dx;

pop bx;

pop ax;

ENDM

В начале макрос создаёт локальные переменные и запоминает значения в регистрах, с которыми он будет работать. В основных строчках используется вложенный цикл:

А) Внешний цикл сравнивает ячейку массива со знаком доллара (изначально буфер данных состоит полностью из этих знаков, так что после занесения строки этот символ будет признаком конца.). В случае совпадения совершается прыжок je endOfString в конец макроса, где восстанавливаются значения в регистрах для дальнейшей работы.

Б) Во внутреннем цикле происходит сравнение текущего символа строки со всеми символами, которые уже были занесены в нашу таблицу частот ранее. Если искомый символ встретится в таблице, то мы совершаем прыжок je found\_c и устанавливаем значение регистра BX в 1; в противном случае при выходе из цикла будет установлено значение 0.

В) Значение регистра BX определяет то, что сделает программа далее: если оно равняется единице, то мы увеличиваем значение частоты искомого символа в таблице на 1; если нулю, то мы увеличиваем значение локальной переменной, отвечающей за длину таблицы, и создаём ячейку таблицы со значениями ([проверяемый символ]; 1).

Г) В самом конце увеличиваем смещение на 1 и повторяем операцию для следующего символа.

Для проверки правильности алгоритма напишем небольшую процедуру по выводу значений массива на экран:

printa MACRO array

local printing, ending, not\_num;

push ax;

push si;

mov si, 0;

mov ah, 02h; код функции

printing:

mov dl, array + si; текущая ячейка

cmp dl, 0; признак конца массива (в основном файле заполнен нулями)

je ending; прыжок за границы бесконечного цикла

cmp dl, 10; проверка на то, выводится ли цифра

jge not\_num;

add dl, 030h; устраняем несоответствие между ASCII-таблицей и значениями массива

not\_num:

int 21h;

mov dl, 20h; "разлепим" ячейки с помощью пробелов

int 21h;

inc si;

jmp printing;

ending:

printc 13;

printc 10;

pop si;

pop ax;

ENDM

Текст основной программы:

INCLUDE \..\Programs\macro.asm

.MODEL SMALL

.STACK 100h

.DATA

buffer DB 512 dup (0)

fileName DB 'Huffman.txt', 0

fileHandle DW ?

\_length DB 255

len DB 12

symbolsArray DB 70 dup (0)

freqArray DB 70 dup (0)

.CODE

START:

; Инициализация сегмента данных и базовая очистка регистров перед началом работы

mov ax, @Data

mov ds, ax

xor ax, ax

xor bx, bx

xor cx, cx

xor dx, dx

xor si, si

xor di, di

openfile fileName

mov fileHandle, ax; после работы макроса в регистре AX находится дескриптор файла

readfile fileHandle, \_length, buffer

prints buffer

freq\_co buffer, symbolsArray, freqArray, len

printa symbolsArray;

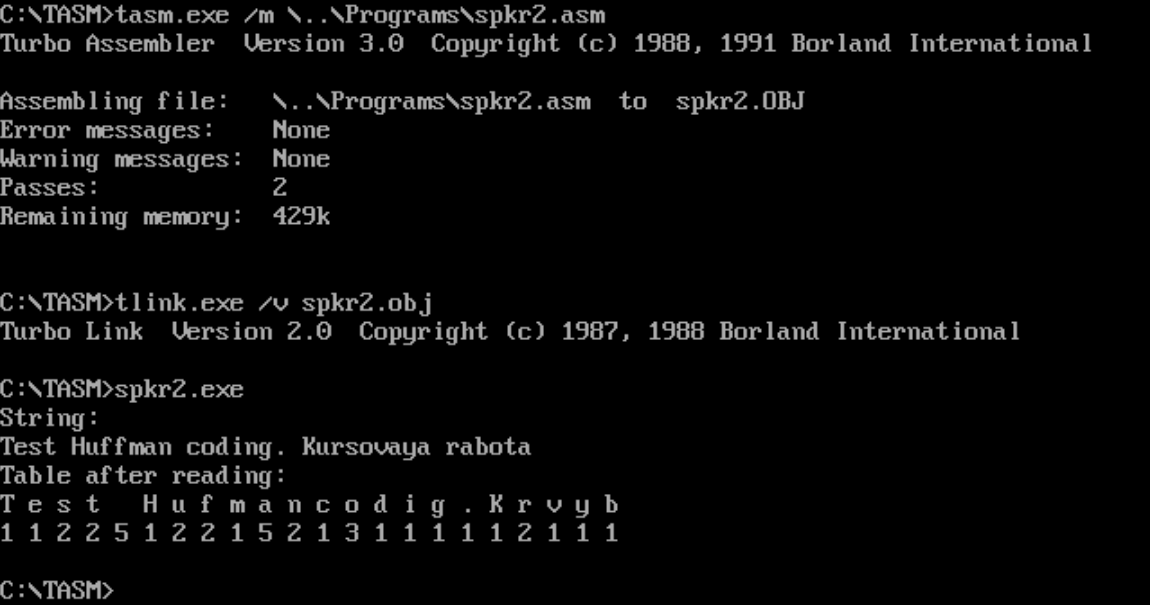
printa freqArray;

; Выход из программы

mov ax, 4c00h;

int 21h;

END START;



Результат корректен.

## Сортировка по частоте встречаемости

Для построения правильного кодового дерева необходима сортировка полученного массива, поскольку так значительно упрощается процесс построения. Для наших целей и сравнительно небольших строк будет достаточно обыкновенной пузырьковой сортировки. Полученный макрос:

sorta MACRO f\_array, c\_array, len

local sort, sortInner, lessEqual;

push ax;

push bx;

push cx;

push dx;

push si;

push di;

mov dl, len;

xor dh, dh;

mov di, dx;

dec di;

sort:

mov si, 0;

sortInner:

mov al, f\_array + si;

mov bl, f\_array + si + 1;

cmp al, bl; сравниваем значения

jle lessEqual; если порядок возрастания не нарушен, то пропускаем перестановку элементов

mov f\_array + si, bl;

mov f\_array + si + 1, al;

; Не забываем про перестановку в массиве с самими символами

mov cl, c\_array + si;

mov dl, c\_array + si + 1;

mov c\_array + si, dl;

mov c\_array + si + 1, cl;

lessEqual:

inc si;

cmp si, di;

jne sortInner;

dec di;

cmp di, 0;

jne sort;

pop di;

pop si;

pop dx;

pop cx;

pop bx;

pop ax;

ENDM

Немного видоизменим основную программу:

INCLUDE \..\Programs\macro.asm

.MODEL SMALL

.STACK 100h

.DATA

stringMessage DB 'String: ', 0

readMessage DB 'Table after reading: ', 0

sortMessage DB 'Sorted table: ', 0

buffer DB 512 dup (0)

fileName DB 'Huffman.txt', 0

fileHandle DW ?

\_length DB 255

len DB 12

symbolsArray DB 70 dup (0)

freqArray DB 70 dup (0)

.CODE

START:

; Инициализация сегмента данных и базовая очистка регистров перед началом работы

mov ax, @Data

mov ds, ax

xor ax, ax

xor bx, bx

xor cx, cx

xor dx, dx

xor si, si

xor di, di

openfile fileName

mov fileHandle, ax; после работы макроса в регистре AX находится дескриптор файла

readfile fileHandle, \_length, buffer

prints stringMessage

prints buffer

freq\_co buffer, symbolsArray, freqArray, len

prints readMessage

printa symbolsArray

printa freqArray

sorta freqArray, symbolsArray, len

prints sortMessage

printa symbolsArray

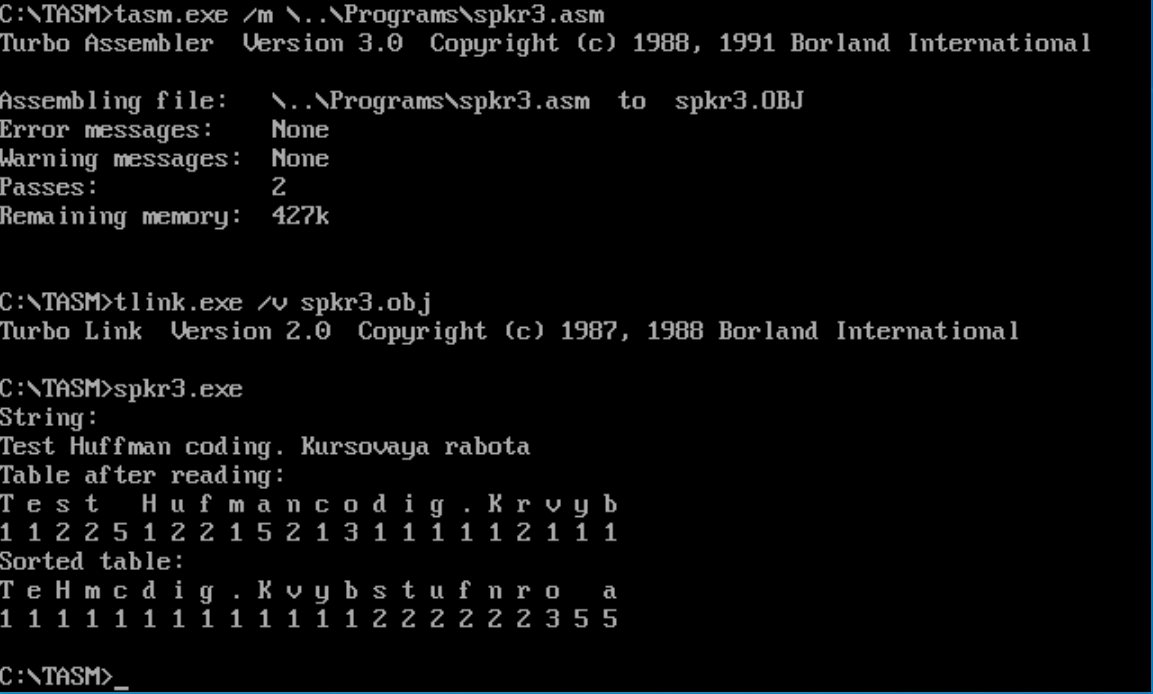
printa freqArray

; Выход из программы

mov ax, 4c00h

int 21h

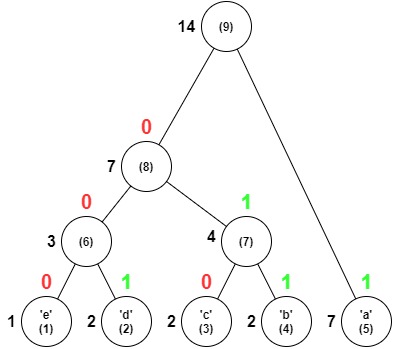
END START;



Результат корректен.

## Построение дерева

На текущий момент мы храним таблицу из двух столбцов – символ и частота встречаемости. Для того, чтобы реализовать алгоритм Хаффмана средствами TASM, присвоим каждому узлу индексы. Индексы нам понадобятся для организации связей (получится своего рода база данных). Индексировать будем в соответствии с простым правилом, изображённом на примере:



Соответственно, исходные узлы получают номера от 1 до N, далее – по порядку появления (всего 2 \* N – 1 узлов).

Помимо индексов самого узла, нужно хранить два индекса, отвечающих за связи. Таким образом, мы получаем следующую структуру данных (по тому же примеру):

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Индекс** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **Символ** | ‘e’ | ‘d’ | ‘c’ | ‘b’ | ‘a’ | ‘’ | ‘’ | ‘’ | ‘’ |
| **Частота** | 1 | 2 | 2 | 2 | 7 | 3 | 4 | 7 | 14 |
| **«Левая связь»** | 0 | 0 | 0 | 0 | 0 | 1 | 3 | 6 | 8 |
| **«Правая связь»** | 0 | 0 | 0 | 0 | 0 | 2 | 4 | 7 | 5 |

В соответствии с таким примером, символы получат следующие коды:

e – 000

d – 001

c – 010

b – 011

a – 1

Начнём с того, что добавим три новых массива той же размерности и длины, и присвоим все необходимые индексы. Вставка в макрос сортировки – присвоение порядковых индексов:

mov dl, len;

xor dh, dh;

mov di, dx;

mov si, 0;

mov bx, 1;

xor ah, ah;

indexing:

mov ax, bx;

mov i\_array + si, al;

inc bx;

inc si;

cmp si, di;

jne indexing;

Макрос вывода массивов индексов на экран:

printindexes MACRO i\_array

local printing, ending;

push ax;

push si;

local num: byte;

mov si, 0;

printing:

mov al, i\_array + si;

mov num, al;

cmp al, 0;

je ending;

printn num; отправляем индекс на вывод

inc si;

jne printing;

ending:

pop si;

pop ax;

ENDM

Макрос по выводу многозначного числа:

printindexes MACRO i\_array, len

local printing, ending;

push ax;

push si;

push di;

local num: byte;

mov al, len;

xor ah, ah;

mov di, ax;

mov si, 0;

printing:

mov al, i\_array + si;

mov num, al;

cmp si, di;

je ending;

printn num; отправляем индекс на вывод

inc si;

jne printing;

ending:

printc 13;

printc 10;

pop di;

pop si;

pop ax;

ENDM

Обновлённая основная программа:

INCLUDE \..\Programs\macro.asm

.MODEL SMALL

.STACK 100h

.DATA

stringMessage DB 'String: ', 0

readMessage DB 'Table after reading: ', 0

sortMessage DB 'Sorted table: ', 0

buffer DB 512 dup (0)

fileName DB 'Huffman.txt', 0

fileHandle DW ?

\_length DB 255

len DB 12

symbolsArray DB 70 dup (0)

freqArray DB 70 dup (0)

indexArray DB 70 dup (0)

leftLinkArray DB 70 dup (0)

rightLinkArray DB 70 dup (0)

.CODE

START:

; Инициализация сегмента данных и базовая очистка регистров перед началом работы

mov ax, @Data

mov ds, ax

xor ax, ax

xor bx, bx

xor cx, cx

xor dx, dx

xor si, si

xor di, di

openfile fileName

mov fileHandle, ax; после работы макроса в регистре AX находится дескриптор файла

readfile fileHandle, \_length, buffer

prints stringMessage

prints buffer

freq\_co buffer, symbolsArray, freqArray, len

prints readMessage

printa symbolsArray

printa freqArray

sorta freqArray, symbolsArray, indexArray, len

prints sortMessage

printa symbolsArray

printa freqArray

printindexes indexArray, len

printindexes leftLinkArray, len

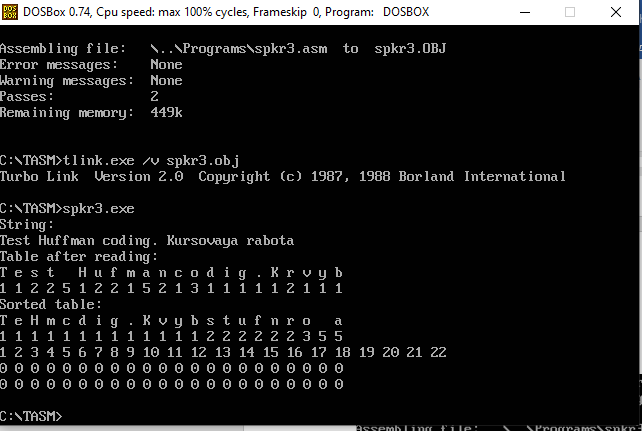
printindexes rightLinkArray, len

; Выход из программы

mov ax, 4c00h

int 21h

END START;

Результат корректен.

Процедура создания новых узлов описана следующей блок-схемой:



Как только значение счётчика переваливает за 2 \* N – 1, цикл прерывается.

Теперь напишем программу для создания дерева:

tree MACRO len, c\_array, f\_array, i\_array, ll\_array, rl\_array

local \_tree, ending;

push ax;

push bx;

push cx;

push dx;

push si;

push di;

local \_len: byte;

; Всего 2n - 1 узлов

mov al, len;

mov \_len, al;

mov bl, 2;

mul bl;

sub al, 1;

mov di, ax; в DI будет храниться конечное число узлов

mov si, 0;

\_tree:

mov al, i\_array + si;

mov bl, i\_array + si + 1;

mov cl, f\_array + si;

mov dl, f\_array + si + 1;

; Запоминаем значения AX и DI

; DI будет временно использоваться для индексации

; Через AX пройдёт конвертация байта локальной переменной \_length в слово для дальнейшей индексации

push di;

push ax;

mov al, \_len;

xor ah, ah;

mov di, ax;

pop ax;

; Заносим связи

; Теперь в AL и BL находятся два индекса

mov ll\_array + di, al;

mov rl\_array + di, bl;

add cl, dl;

; Заносим частоту

; Последняя ячейка - по адресу "длина - 1", так что наш новый узел располагается по адресу "длина"

; В CL хранится сумма частот

mov f\_array + di, cl;

inc di; стало на 1 ячейку больше

mov cx, di;

; Заносим новый индекс

; В CL хранится новое значение длины

mov i\_array + di - 1, cl;

; Возвращаем в локальную переменную новое, увеличенное значение длины

mov ax, di;

mov \_len, al;

pop di;

; Осталось правильно расположить элемент в таблице

sortnewelement c\_array, f\_array, i\_array, ll\_array, rl\_array, \_len

add si, 2;

cmp si, di;

jge ending;

jmp \_tree;

ending:

mov dx, di;

mov len, dl;

pop di;

pop si;

pop dx;

pop cx;

pop bx;

pop ax;

ENDM

Сортировка нового элемента:

sortnewelement MACRO c\_array, f\_array, i\_array, ll\_array, rl\_array, new\_len

local sortInner, lessEqual;

push ax;

push bx;

push si;

push di;

mov al, new\_len;

xor ah, ah;

mov si, ax;

; Процедура практически скопирована из sorta

; за исключением того, что сделать достаточно одного цикла, который будет перемещать наш новый элемент

sortInner:

mov al, f\_array + si - 2;

mov bl, f\_array + si - 1;

cmp al, bl; сравниваем значения

jle lessEqual; если порядок возрастания не нарушен, то пропускаем перестановку элементов

mov f\_array + si - 2, bl;

mov f\_array + si - 1, al;

; Не забываем про перестановку в остальных строках таблицы

mov cl, c\_array + si - 2;

mov dl, c\_array + si - 1;

mov c\_array + si - 2, dl;

mov c\_array + si - 1, cl;

mov cl, i\_array + si - 2;

mov dl, i\_array + si - 1;

mov i\_array + si - 2, dl;

mov i\_array + si - 1, cl;

mov cl, ll\_array + si - 2;

mov dl, ll\_array + si - 1;

mov ll\_array + si - 2, dl;

mov ll\_array + si - 1, cl;

mov cl, rl\_array + si - 2;

mov dl, rl\_array + si - 1;

mov rl\_array + si - 2, dl;

mov rl\_array + si - 1, cl;

lessEqual:

dec si;

cmp si, 1;

jne sortInner;

pop di;

pop si;

pop bx;

pop ax;

ENDM

Макрос по выводу дерева на экран:

printtree MACRO len, c\_array, f\_array, i\_array, ll\_array, rl\_array

local printing, ending, not\_space\_i, not\_space\_f, not\_space\_l;

push ax;

push si;

push di;

local argument: byte;

mov al, len;

xor ah, ah;

mov di, ax;

mov si, 0;

printing:

printc 20h;

mov al, i\_array + si;

mov argument, al;

printn argument;

; Для форматирования вывода добавим дополнительный пробел после однозначных чисел

cmp al, 9;

jg not\_space\_i;

printc 20h;

not\_space\_i:

printc 20h;

mov al, c\_array + si;

mov argument, al;

printc argument;

printc 20h;

mov al, f\_array + si;

mov argument, al;

printn argument;

; Для форматирования вывода добавим дополнительный пробел после однозначных чисел

cmp al, 9;

jg not\_space\_f;

printc 20h;

not\_space\_f:

printc 20h;

mov al, ll\_array + si;

mov argument, al;

printn argument;

; Для форматирования вывода добавим дополнительный пробел после однозначных чисел

cmp al, 9;

jg not\_space\_l;

printc 20h;

not\_space\_l:

printc 20h;

mov al, rl\_array + si;

mov argument, al;

printn argument;

printc 13;

printc 10;

inc si;

cmp si, di;

je ending;

jmp printing;

ending:

pop di;

pop si;

pop ax;

ENDM

Видоизменённая основная программа:

INCLUDE \..\Programs\macro.asm

.MODEL SMALL

.STACK 100h

.DATA

buffer DB 512 dup (0)

fileName DB 'Huffman.txt', 0

fileHandle DW ?

\_length DB 255

len DB 10

symbolsArray DB 70 dup (0)

freqArray DB 70 dup (0)

indexArray DB 70 dup (0)

leftLinkArray DB 70 dup (0)

rightLinkArray DB 70 dup (0)

.CODE

START:

; Инициализация сегмента данных и базовая очистка регистров перед началом работы

mov ax, @Data

mov ds, ax

xor ax, ax

xor bx, bx

xor cx, cx

xor dx, dx

xor si, si

xor di, di

openfile fileName

mov fileHandle, ax; после работы макроса в регистре AX находится дескриптор файла\

readfile fileHandle, \_length, buffer

freq\_co buffer, symbolsArray, freqArray, len

sorta freqArray, symbolsArray, indexArray, len

tree len, symbolsArray, freqArray, indexArray, leftLinkArray, rightLinkArray

printtree len, symbolsArray, freqArray, indexArray, leftLinkArray, rightLinkArray

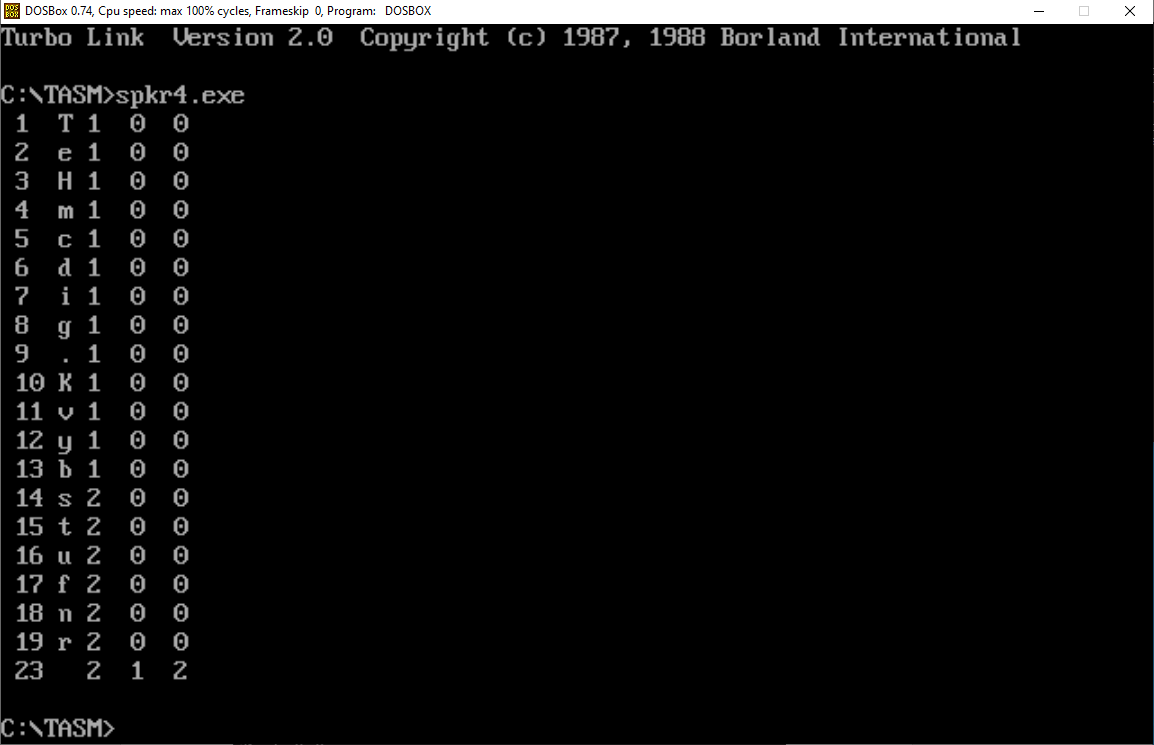
; Выход из программы

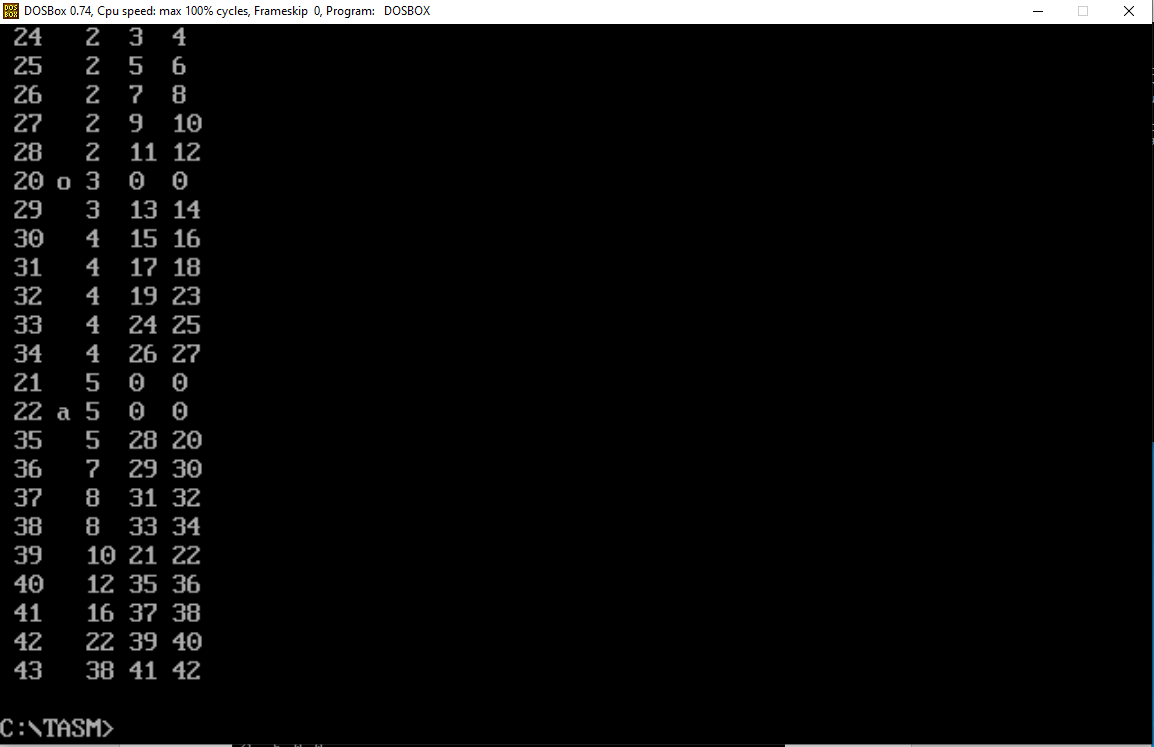
mov ax, 4c00h

int 21h

END START;

Вывод на консоль в формате [Индекс][Символ][Частота][Левая связь][Правая связь]:





Для проверки запишем вручную коды всех символов по полученному дереву:

T – **00110**

e – **00111**

H – **01000**

m – **01001**

c – **01010**

d – **01011**

i – **01100**

g – **01101**

. – **01110**

K – **01111**

v – **11000**

y – **11001**

b – **11100**

s – **11101**

t – **11110**

u – **11111**

f – **0000**

n **– 0001**

r – **0010**

o – **1101**

[SPACE] – **100**

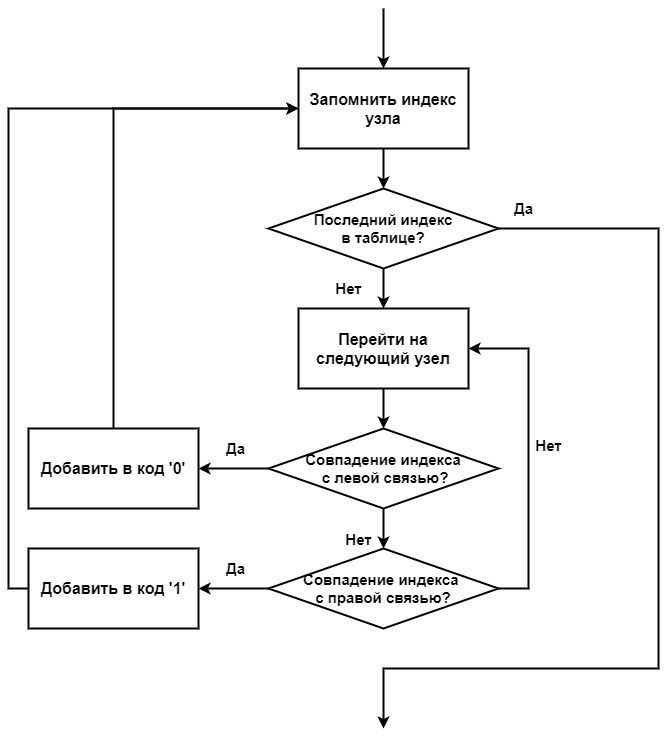
a – **101**

Результат корректен.

## Получение кодировки

Теперь нужно совершить проход по дереву и собрать код для каждого символа. Было бы удобно задать двумерный массив, но TASM не позволяет сделать этого напрямую. Вместо этого мы сделаем одномерный массив размерностью “8 \* [длина]” для хранения всех информационных битов (первые 8 элементов массива – для первого элемента, и так далее). Будем обращаться к необходимым ячейкам через два индексных регистра – SI и DI – и тогда для обращения к биту DI ячейки SI получим выражение [SI \* 8 + DI].

Нам понадобится пройтись по каждому узлу с символом и провести следующий анализ:



Макрос по созданию кодов:

huffcodes MACRO len, c\_array, code\_array, i\_array, ll\_array, rl\_array

local coding, codingInner, continue, left\_found, left\_next, right\_found, right\_next;

push ax;

push bx;

push cx;

push dx;

push si;

push di;

xor ch, ch;

mov cl, len;

mov si, 0;

coding:

mov al, c\_array + si;

cmp al, 0;

; Если символ отсутствует, то узел вторичный и создавать ему код мы не будем

je continue;

; Если символ присутствует, то заносим индекс вместо;

; сам символ нам на текущий момент безразличен, поскольку мы знаем его номер в массиве

; и сможем индексировать в массиве кодов

mov al, i\_array + si;

; Проверка начинается с узла, следующего за текущим

mov di, si;

inc di;

; В DX будет храниться счётчик количества символов;

; в нужный момент мы перенесём его в DI (сохранив последний в стеке)

; и занесём 0 или 1 в соответствующую ячейку

mov dx, 0;

codingInner:

;Проверяем левую связь

mov bl, ll\_array + di;

cmp bl, al;

; Если не совпадение, то проверяем правую

; Если совпадение, то добавляем в необходимую ячейку массива ноль

jne left\_next;

; Теперь нашим текущим индексом становится тот узел, в котором мы обнаружили связь

mov al, i\_array + di;

; Запомним значение DI и занесём в него номер ячейки

; Через AX пройдёт умножение на 8

push di;

push ax;

push dx;

mov ax, si;

mov bl, 8;

mul bl;

add ax, dx;

; AX = SI \* 8 + DX

mov di, ax;

mov code\_array + di, 30h;

inc di;

pop dx;

pop ax;

pop di;

; Смещаемся на одну ячейку

inc dx;

left\_next:

; Проверяем правую связь

mov bl, rl\_array + di;

cmp bl, al;

; Если не совпадение, то переходим к сравнению со следующим узлом

; Если совпадение, то добавляем единичку

jne right\_next;

; Теперь нашим текущим индексом становится тот узел, в котором мы обнаружили связь

mov al, i\_array + di;

; Запомним значение DI и занесём в него значение счётчика

push di;

push ax;

push dx;

mov ax, si;

mov bl, 8;

mul bl;

add ax, dx;

; AX = SI \* 8 + DX

mov di, ax;

mov code\_array + di, 31h;

inc di;

pop dx;

pop ax;

pop di;

; Смещаемся на одну ячейку

inc dx;

right\_next:

; До конца таблицы

inc di;

cmp di, cx;

jne codingInner;

continue:

; До конца таблицы

inc si;

cmp si, cx;

jne coding;

pop di;

pop si;

pop dx;

pop cx;

pop bx;

pop ax;

ENDM

Макрос для выведения кодов на консоль:

printcodes MACRO c\_array, code\_array, len

local printing, codes, continue, ending;

push ax;

push bx;

push si;

push di;

local argument: byte;

mov bl, len;

xor bh, bh;

mov si, 0;

printing:

mov al, c\_array + si;

cmp al, 0;

je continue;

mov argument, al;

printc argument;

printc 20h;

mov di, 0;

codes:

push di;

push bx;

mov ax, si;

mov bl, 8;

mul bl;

add ax, di;

mov di, ax;

mov al, code\_array + di;

mov argument, al;

printc argument;

pop bx;

pop di;

inc di;

cmp di, 8;

jne codes;

printc 13;

printc 10;

continue:

inc si;td

cmp si, bx;

je ending;

jmp printing;

ending:

pop di;

pop si;

pop ax;

ENDM

Основная программа:

INCLUDE \..\Programs\macro.asm

.MODEL SMALL

.STACK 100h

.DATA

buffer DB 512 dup (0)

fileName DB 'Huffman.txt', 0

fileHandle DW ?

\_length DB 255

len DB 10

symbolsArray DB 70 dup (0)

freqArray DB 70 dup (0)

indexArray DB 70 dup (0)

leftLinkArray DB 70 dup (0)

rightLinkArray DB 70 dup (0)

codeArray DB 560 dup (0)

.CODE

START:

; Инициализация сегмента данных и базовая очистка регистров перед началом работы

mov ax, @Data

mov ds, ax

xor ax, ax

xor bx, bx

xor cx, cx

xor dx, dx

xor si, si

xor di, di

openfile fileName

mov fileHandle, ax; после работы макроса в регистре AX находится дескриптор файла\

readfile fileHandle, \_length, buffer

freq\_co buffer, symbolsArray, freqArray, len

sorta freqArray, symbolsArray, indexArray, len

tree len, symbolsArray, freqArray, indexArray, leftLinkArray, rightLinkArray

huffcodes len, symbolsArray, codeArray, indexArray, leftLinkArray, rightLinkArray

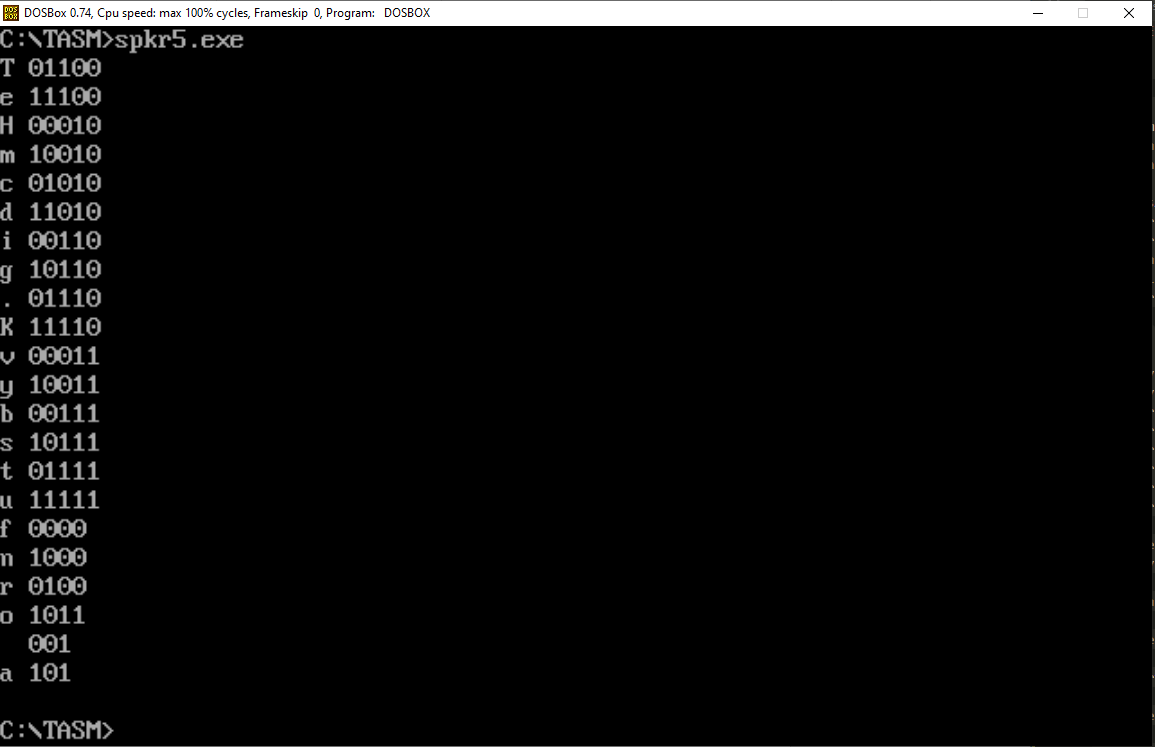
printcodes symbolsArray, codeArray, len

; Выход из программы

mov ax, 4c00h

int 21h

END START;



Результат корректен, но мы не учли того, что двигались от листов с символами к корню, в то время как корректный код получается при движении в обратном направлении. Добавим в конец нашего макроса по построению кодов процедуру разворота:

; Здесь же разворачиваем битовую последовательность в другую сторону

mov si, 0;

reverse:

xor ah, ah;

push si;

mov ax, si;

mov bl, 8;

mul bl;

mov di, ax;

add al, 8;

mov si, ax;

mov dl, 4;

turn:

dec si;

; Нужно учитывать факт того, что в наших восьми ячейках содержится и пустой код

; Создадим поверх обмена дополнительную проверку

mov al, code\_array + si;

; Достаточно проверить только AL, поскольку в нём - символы с конца

cmp al, 0;

je null\_char;

mov bl, code\_array + di;

mov code\_array + di, al;

mov code\_array + si, bl;

inc di;

null\_char:

dec dl;

cmp si, di;

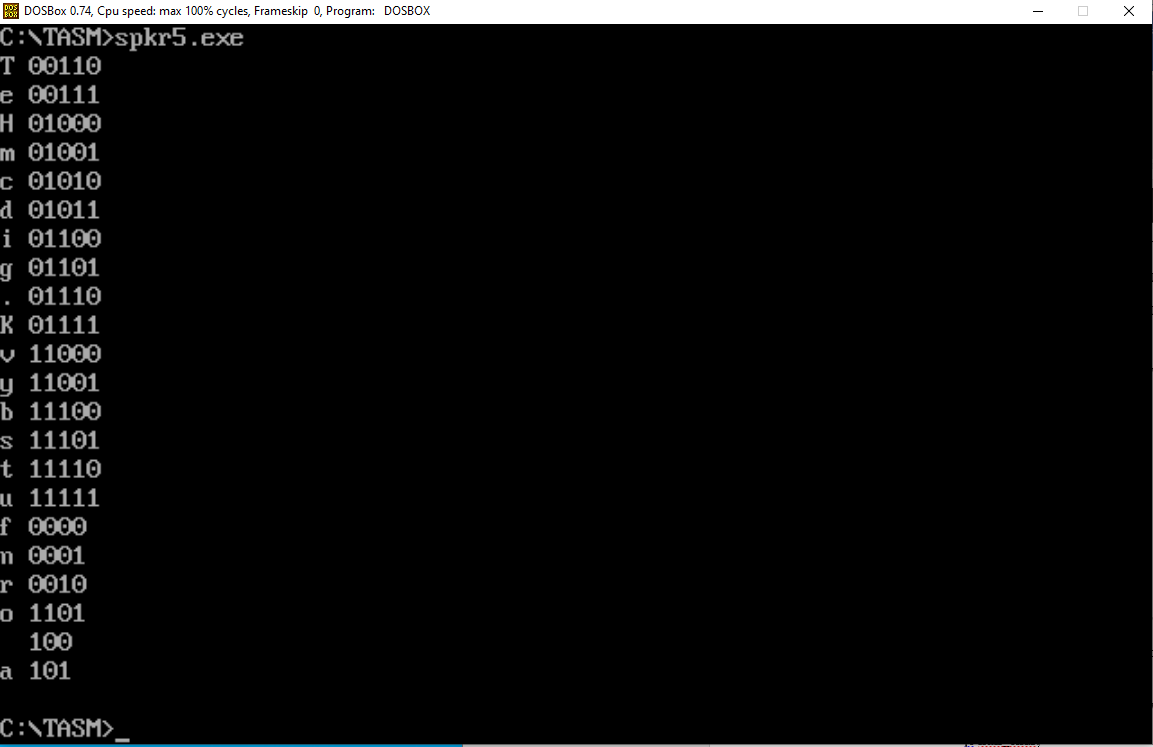
jg turn;

pop si;

inc si;

cmp si, cx;

jne reverse;



Результат корректен и полностью соответствует ручной проверке в четвёртом пункте.

## Кодирование исходной строки

Создадим внутри программы два файла: в один файл мы поместим кодовую таблицу, во второй – закодированную строку. Первый файл нам пригодится для второй части задания – декодирования, - а второй будет результатом первой.

Макрос по записи кодовой таблицы:

writefile\_cp MACRO handle, c\_array, code\_array, len, enter\_

local writing, writing\_inner, continue, null\_char;

push ax;

push bx;

push cx;

push dx;

push si;

push di;

xor ax, ax;

mov ax, len;

mov di, ax;

mov si, 0

writing:

; Признак конца таблицы

cmp c\_array + si, 0;

je continue;

; Пишем символ

mov ah, 40h;

mov bx, handle;

mov cx, 1;

lea dx, c\_array + si;

int 21h;

push di;

push si;

mov ax, si;

mov bl, 8;

mul bl;

mov si, ax;

mov di, 0;

; Пишем коды

writing\_inner:

cmp code\_array + si, 0;

je null\_char;

mov ah, 40h;

mov bx, handle;

mov cx, 1;

lea dx, code\_array + si;

int 21h;

null\_char:

inc si;

inc di;

cmp di, 8;

jne writing\_inner;

pop si;

pop di;

; Пишем перенос строки

mov ah, 40h;

mov bx, handle;

mov cx, 2;

lea dx, enter\_;

int 21h;

continue:

inc si;

cmp si, di;

jne writing;

pop di;

pop si;

pop dx;

pop cx;

pop bx;

pop ax;

ENDM

Макрос по кодированию строки:

writefile\_encode MACRO handle, string, c\_array, code\_array, enter\_, len

local encoding, searching, continue, not\_found, null\_char, coding\_end, encoding\_inner, ending;

push ax;

push bx;

push si;

push di;

mov si, 0;

encoding:

mov ah, string + si;

; Признак конца строки

cmp ah, 0;

je ending;

; Запоминаем SI

push si;

; Ищем символ в кодовой таблице

mov si, 0;

searching:

mov al, c\_array + si;

cmp ah, al;

jne not\_found;

; AX с символами нам больше не нужен, поскольку мы нашли совпадение и можем вычислить все индексы

mov ax, si;

mov bl, 8;

mul bl;

mov si, ax;

mov di, 0;

; Записываем кодовую последовательность

encoding\_inner:

cmp code\_array + si, 0

je continue;

mov ah, 40h;

mov bx, handle;

mov cx, 1;

lea dx, code\_array + si;

int 21h;

inc di;

inc si;

cmp di, 8;

je continue;

jmp encoding\_inner;

not\_found:

inc si;

cmp si, len;

je continue;

jmp searching;

continue:

pop si;

inc si;

jmp encoding;

ending:

pop di;

pop si;

pop bx;

pop ax;

ENDM

Основная программа:

INCLUDE \..\Programs\macro.asm

INCLUDE \..\Programs\print\_m.asm

.MODEL SMALL

.STACK 100h

.DATA

buffer DB 512 dup (0)

fileName DB 'Huffman.txt', 0

fileHandle DW ?

newFileName DB 'Encoding.txt', 0

newFileHandle DW ?

cpFileName DB 'cp.txt', 0

cpFileHandle DW ?

\_length DB 255

len DW 10

symbolsArray DB 120 dup (0)

freqArray DB 120 dup (0)

indexArray DB 120 dup (0)

leftLinkArray DB 120 dup (0)

rightLinkArray DB 120 dup (0)

codeArray DB 960 dup (0)

enter\_ DB 13, 10

.CODE

START:

; Инициализация сегмента данных и базовая очистка регистров перед началом работы

mov ax, @Data

mov ds, ax

xor ax, ax

xor bx, bx

xor cx, cx

xor dx, dx

xor si, si

xor di, di

openfile fileName

mov fileHandle, ax; после работы макроса в регистре AX находится дескриптор файла\

readfile fileHandle, \_length, buffer

freq\_co buffer, symbolsArray, freqArray, len

sorta freqArray, symbolsArray, indexArray, len

tree len, symbolsArray, freqArray, indexArray, leftLinkArray, rightLinkArray

huffcodes len, symbolsArray, codeArray, indexArray, leftLinkArray, rightLinkArray

createfile cpFileName

mov cpFileHandle, ax;

createfile newFileName

mov newFileHandle, ax;

writefile\_cp cpFileHandle, symbolsArray, codeArray, len, enter\_

writefile\_encode newFileHandle, buffer, symbolsArray, codeArray, enter\_, len

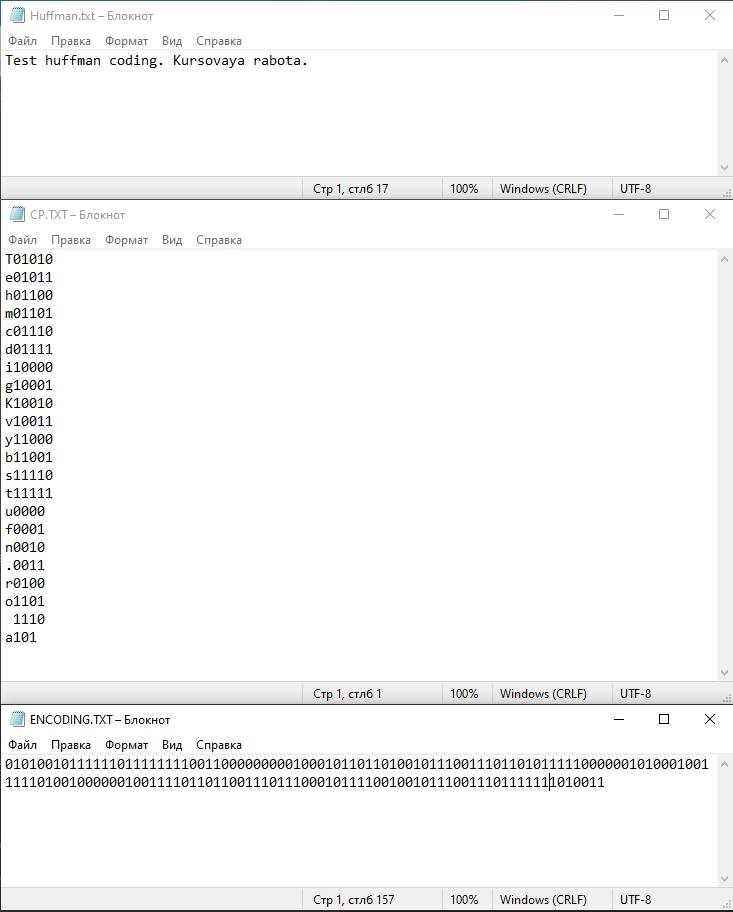
; Выход из программы

mov ax, 4c00h

int 21h

END START;

Результат работы:



Результат полностью верен, алгоритм Хаффмана реализован.

# **Декодирование**

Процесс декодирования значительно проще: нужно получить кодовую таблицу из одного файла, закодированную строку из другого и записать корректный результат в третий.

Макрос считывания кодовой таблицы:

getcodes MACRO buffer, c\_array, code\_array, len

local codes, coding, next, end\_of\_code, ending, continue;

push ax;

push cx;

push si;

push di;

local char\_length: word;

mov char\_length, 0;

mov si, 0;

mov di, 0;

codes:

mov ah, buffer + si;

; Признак конца буфера

cmp ah, 0;

je ending;

inc char\_length;

mov c\_array + di, ah;

; Запоминаем DI и начинаем обращение к массиву кодов

push di;

mov ax, di;

mov bl, 8;

mul bl;

mov di, ax;

mov cx, 0;

; Начинаем с символа, следующего за буквой (в файле они "слеплены")

; Заносим все 8

coding:

inc si;

mov ah, buffer + si;

mov code\_array + di, ah;

inc di;

inc cx;

cmp cx, 8;

jne coding;

pop di;

inc di;

; Проскакиваем связку 13-10

add si, 3;

jmp codes;

ending:

mov ax, char\_length;

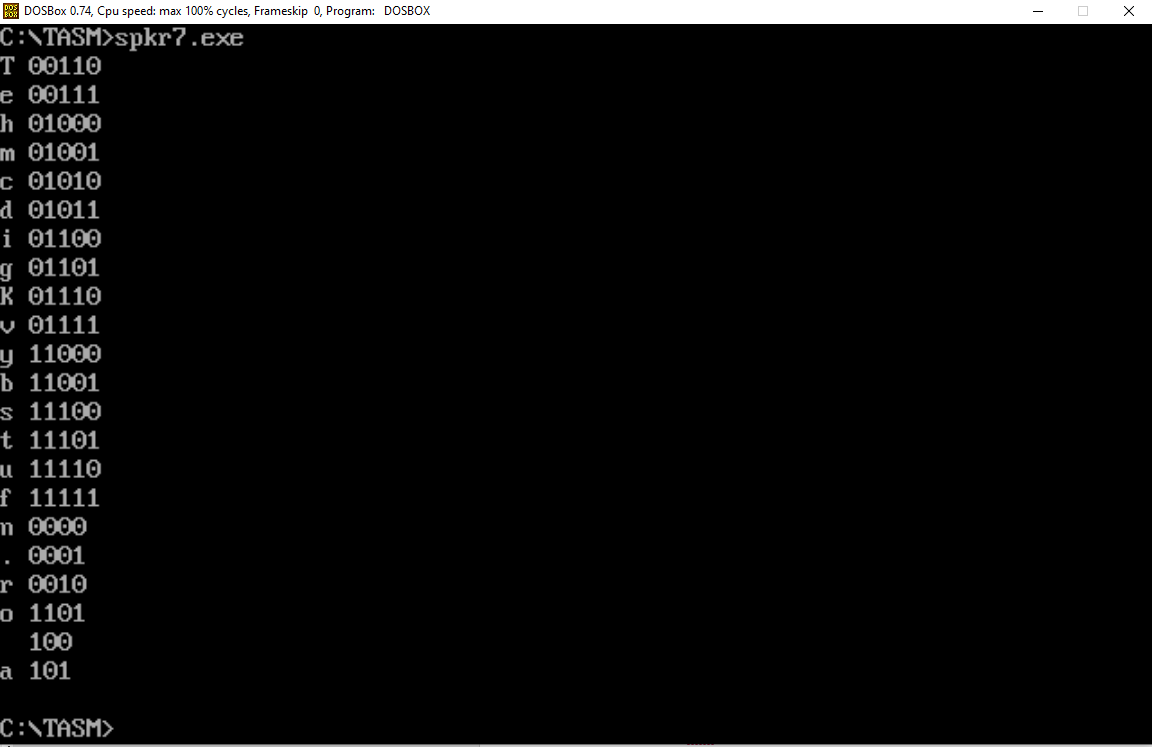
mov len, ax;

pop di;

pop si;

pop ax;

ENDM



Считывание кодов прошло без ошибок.

Макрос по записи раскодированной строки в файл:

decodestring MACRO buffer, c\_array, code\_array, r\_buffer, handle, len

local decoding, ending, decoding\_inner, not\_equal, found, found\_continue, next\_char, next, self\_clear, continue, checking;

push ax;

push bx;

push si;

push di;

local code\_length: word;

mov ax, len;

mov bl, 8;

mul bl;

mov code\_length, ax;

mov si, 0;

mov di, 0;

decoding:

mov ah, buffer + si;

cmp ah, 0;

je ending;

; Будем записывать в буфер каждый новый символ и проверять на совпадение со всей считанной из файла таблицей

; Если найдём совпадение, то отправляем в файл символ, обнуляем буфер и продолжаем

mov r\_buffer + di, ah;

; Сверяем содержимое восьмибайтного буфера со всей кодовой таблицей

; В DI будет кол-во символов в буфере, в SI - текущее положение

push si;

push di;

mov si, 0;

mov di, 0;

; Проходим по всей кодовой таблице

decoding\_inner:

push si;

add si, di;

mov ah, code\_array + si;

pop si;

cmp ah, r\_buffer + di;

; Если несовпадение, то переходим к следующему символу

jne next\_char;

; Если совпадение, то продолжаем сравнивать

inc di;

cmp di, 8;

; Если мы прошли восемь ячеек и таким образом получили полное совпадение

je found;

; Иначе просто продолжаем

jmp next;

found:

; Надо поделить SI на 8 из-за того, что мы индексировали с его помощью кодовый массив

mov ax, si;

mov bl, 8;

div bl;

mov si, ax;

mov ah, 40h;

mov bx, handle;

mov cx, 1;

lea dx, c\_array + si;

int 21h;

; Теперь надо очистить буфер

mov si, 0;

self\_clear:

mov r\_buffer + si, 0;

inc si;

cmp si, 8;

jne self\_clear;

pop di;

mov di, 0;

; Если мы нашли совпадение по символу, то нам требуется не inc di, а обнуление,

; поскольку мы уже очистили буфер и нужно начинать запись заново

jmp found\_continue;

next\_char:

; Если мы получили несовпадение, то надо обнулить DI, а SI добавить 8 (перейти к следующему символу)

mov di, 0;

add si, 8;

next:

; Сравниваем с концом кодового массива

cmp si, code\_length;

jge continue;

jmp decoding\_inner;

continue:

pop di;

inc di;

found\_continue:

pop si;

inc si;

jmp decoding;

ending:

pop di;

pop si;

pop bx;

pop ax;

ENDM

Основная программа:

INCLUDE \..\Programs\macro.asm

INCLUDE \..\Programs\print\_m.asm

.MODEL SMALL

.STACK 100h

.DATA

buffer DB 1024 dup (0)

destFileName DB 'Decoding.txt', 0

destFileHandle DW ?

sourceFileName DB 'Encoding.txt', 0

sourceFileHandle DW ?

cpFileName DB 'cp.txt', 0

cpFileHandle DW ?

\_length DW 1024

len DW 10

symbolsArray DB 120 dup (0)

codeArray DB 960 dup (0)

enter\_ DB 13, 10

readBuffer DB 8 dup (0)

.CODE

START:

; Инициализация сегмента данных и базовая очистка регистров перед началом работы

mov ax, @Data

mov ds, ax

xor ax, ax

xor bx, bx

xor cx, cx

xor dx, dx

xor si, si

xor di, di

openfile cpFileName

mov cpFileHandle, ax;

openfile sourceFileName

mov sourceFileHandle, ax;

createfile destFileName

mov destFileHandle, ax;

readfile cpFileHandle, \_length, buffer

getcodes buffer, symbolsArray, codeArray, len

readfile sourceFileHandle, \_length, buffer

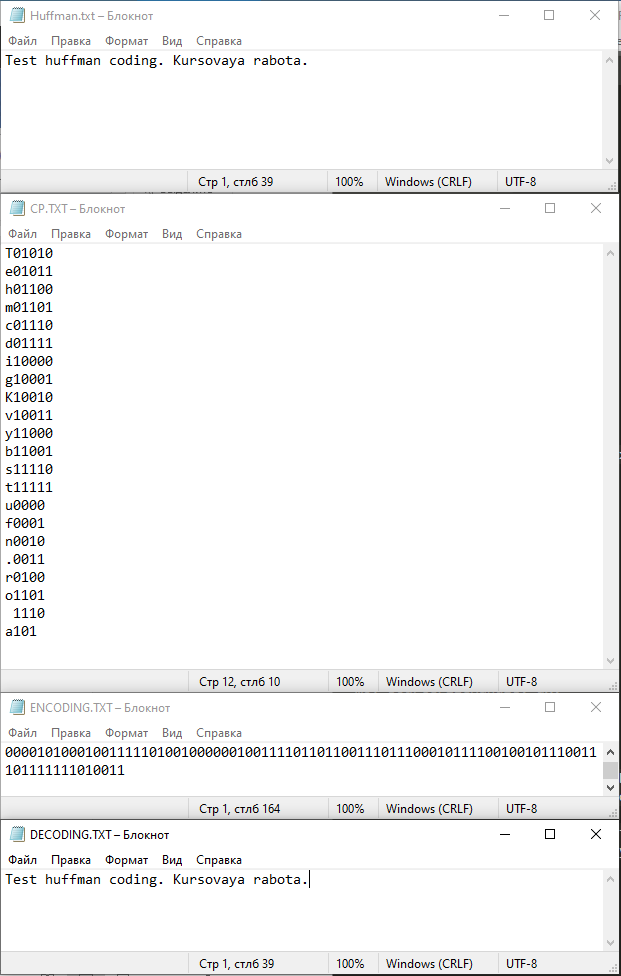
decodestring buffer, symbolsArray, codeArray, readBuffer, destFileHandle, len

; Выход из программы

mov ax, 4c00h

int 21h

END START;



Результат работы алгоритма в обе стороны полностью верен.

# **Примечания**

Ссылка на файлы с программным кодом:

<https://github.com/Zaburunier/SP_coursework>

# **Полный листинг программы**

# **Файл macro.asm**

В данном файле находятся все макросы, связанные непосредственно с алгоритмом:

openfile MACRO filename

push dx; сохраняем параметр, находившиеся в регистре до вызова функции

mov ax, 3d00h; код функции

mov dx, offset filename; параметры функции

int 21h; вызов прерывания

pop dx; восстанавливаем параметр

ENDM

readfile MACRO handle, count, buffer

push ax;

push bx;

push cx;

push dx;

mov ah, 3fh;

mov bx, handle;

mov cx, count;

mov dx, offset buffer;

int 21h;

pop dx;

pop cx;

pop bx;

pop ax;

ENDM

createfile MACRO filename

push cx;

push dx;

mov ax, 3c00h;

mov cx, 0;

mov dx, offset filename;

int 21h;

pop dx;

pop cx;

ENDM

freq\_co MACRO string, c\_array, f\_array, len

local freqCount, endOfString, found, not\_found, cont, checking, found\_c, nextStep; локальные метки

push ax;

push bx;

push dx;

push di;

push si;

push bp;

local c\_length: word, char: byte; локальные переменные

mov c\_length, 1;

mov si, 0;

freqCount:

mov ah, string + si;

cmp ah, 0; признак конца строки - код символа знака доллара

je endOfString; если конец строки, то цикл из безусловных прыжков прерывается

; Сверяем со всеми находящимися в массиве символами

mov char, ah;

mov di, 0;

mov dx, c\_length;

checking:

mov ah, c\_array + di;

cmp ah, char;

je found\_c; совершаем прыжок, если нашлось сравнение

inc di;

cmp di, dx;

jne checking;

mov bx, 0;

jmp nextStep;

found\_c:

mov bx, 1;

nextStep:

; На основе значения BX совершаем дальнейшие действия

cmp bx, 1;

je found;

cmp bx, 0;

je not\_found;

found:

inc f\_array + di; увеличиваем частоту символа на 1

jmp cont;

not\_found:

inc c\_length; новый символ

mov di, dx; в DL сидит c\_length до работы инкрементора

mov dl, char;

mov c\_array + di - 1, dl; заносим символ в массив

mov f\_array + di - 1, 1; и частоту

jmp cont;

cont:

inc si;

jmp freqCount;

endOfString:

; Запоминаем длину для будущих операций

mov ax, c\_length;

dec ax;

mov len, ax;

pop bp;

pop si;

pop di;

pop dx;

pop bx;

pop ax;

ENDM

sorta MACRO f\_array, c\_array, i\_array, len

local sort, sortInner, lessEqual, indexing;

push ax;

push bx;

push cx;

push dx;

push si;

push di;

mov di, len;

dec di;

sort:

mov si, 0;

sortInner:

mov al, f\_array + si ;

mov bl, f\_array + si + 1;

cmp al, bl; сравниваем значения

jle lessEqual; если порядок возрастания не нарушен, то пропускаем перестановку элементов

mov f\_array + si, bl;

mov f\_array + si + 1, al;

; Не забываем про перестановку в массиве с самими символами

mov cl, c\_array + si;

mov dl, c\_array + si + 1;

mov c\_array + si, dl;

mov c\_array + si + 1, cl;

lessEqual:

inc si;

cmp si, di;

jne sortInner;

dec di;

cmp di, 0;

jne sort;

; В этом же макросе добавим индексы отсортированным узлам

mov di, len;

mov si, 0;

mov bx, 1;

xor ah, ah;

indexing:

mov al, bl;

mov i\_array + si, al;

inc bl;

inc si;

cmp si, di;

jne indexing;

pop di;

pop si;

pop dx;

pop cx;

pop bx;

pop ax;

ENDM

tree MACRO len, c\_array, f\_array, i\_array, ll\_array, rl\_array

local \_tree, ending;

push ax;

push bx;

push cx;

push dx;

push si;

push di;

local \_len: word;

; Всего 2n - 1 узлов

mov ax, len;

mov \_len, ax;

mov bl, 2;

mul bl;

sub al, 1;

mov di, ax; в DI будет храниться конечное число узлов

mov si, 0;

\_tree:

mov al, i\_array + si;

mov bl, i\_array + si + 1;

mov cl, f\_array + si;

mov dl, f\_array + si + 1;

; Запоминаем значения AX и DI

; DI будет временно использоваться для индексации

; Через AX пройдёт конвертация байта локальной переменной \_length в слово для дальнейшей индексации

push di;

push ax;

mov ax, \_len;

xor ah, ah;

mov di, ax;

pop ax;

; Заносим связи

; Теперь в AL и BL находятся два индекса

mov ll\_array + di, al;

mov rl\_array + di, bl;

add cl, dl;

; Заносим частоту

; Последняя ячейка - по адресу "длина - 1", так что наш новый узел располагается по адресу "длина"

; В CL хранится сумма частот

mov f\_array + di, cl;

inc di; стало на 1 ячейку больше

mov cx, di;

; Заносим новый индекс

; В CL хранится новое значение длины

mov i\_array + di - 1, cl;

; Возвращаем в локальную переменную новое, увеличенное значение длины

mov ax, di;

mov \_len, ax;

pop di;

; Осталось правильно расположить элемент в таблице

sortnewelement c\_array, f\_array, i\_array, ll\_array, rl\_array, \_len

add si, 2;

cmp si, di;

jge ending;

jmp \_tree;

ending:

mov dx, di;

mov len, dx;

pop di;

pop si;

pop dx;

pop cx;

pop bx;

pop ax;

ENDM

sortnewelement MACRO c\_array, f\_array, i\_array, ll\_array, rl\_array, new\_len

local sortInner, lessEqual;

push ax;

push bx;

push si;

push di;

mov ax, new\_len;

mov si, ax;

; Процедура практически скопирована из sorta

; за исключением того, что сделать достаточно одного цикла, который будет перемещать наш новый элемент

sortInner:

mov al, f\_array + si - 2;

mov bl, f\_array + si - 1;

cmp al, bl; сравниваем значения

jle lessEqual; если порядок возрастания не нарушен, то пропускаем перестановку элементов

mov f\_array + si - 2, bl;

mov f\_array + si - 1, al;

; Не забываем про перестановку в остальных строках таблицы

mov cl, c\_array + si - 2;

mov dl, c\_array + si - 1;

mov c\_array + si - 2, dl;

mov c\_array + si - 1, cl;

mov cl, i\_array + si - 2;

mov dl, i\_array + si - 1;

mov i\_array + si - 2, dl;

mov i\_array + si - 1, cl;

mov cl, ll\_array + si - 2;

mov dl, ll\_array + si - 1;

mov ll\_array + si - 2, dl;

mov ll\_array + si - 1, cl;

mov cl, rl\_array + si - 2;

mov dl, rl\_array + si - 1;

mov rl\_array + si - 2, dl;

mov rl\_array + si - 1, cl;

lessEqual:

dec si;

cmp si, 1;

jne sortInner;

pop di;

pop si;

pop bx;

pop ax;

ENDM

huffcodes MACRO len, c\_array, code\_array, i\_array, ll\_array, rl\_array

local coding, codingInner, continue, left\_found, left\_next, right\_found, right\_next, reverse, turn, null\_char;

push ax;

push bx;

push cx;

push dx;

push si;

push di;

local first: byte, last: byte;

xor ch, ch;

mov cx, len;

mov si, 0;

coding:

mov al, c\_array + si;

cmp al, 0;

; Если символ отсутствует, то узел вторичный и создавать ему код мы не будем

je continue;

; Если символ присутствует, то заносим индекс вместо;

; сам символ нам на текущий момент безразличен, поскольку мы знаем его номер в массиве

; и сможем индексировать в массиве кодов

mov al, i\_array + si;

; Проверка начинается с узла, следующего за текущим

mov di, si;

inc di;

; В DX будет храниться счётчик количества символов;

; в нужный момент мы перенесём его в DI (сохранив последний в стеке)

; и занесём 0 или 1 в соответствующую ячейку

mov dx, 0;

codingInner:

;Проверяем левую связь

mov bl, ll\_array + di;

cmp bl, al;

; Если не совпадение, то проверяем правую

; Если совпадение, то добавляем в необходимую ячейку массива ноль

jne left\_next;

; Теперь нашим текущим индексом становится тот узел, в котором мы обнаружили связь

mov al, i\_array + di;

; Запомним значение DI и занесём в него номер ячейки

; Через AX пройдёт умножение на 8

push di;

push ax;

push dx;

mov ax, si;

mov bl, 8;

mul bl;

add ax, dx;

; AX = SI \* 8 + DX

mov di, ax;

mov code\_array + di, 30h;

inc di;

pop dx;

pop ax;

pop di;

; Смещаемся на одну ячейку

inc dx;

left\_next:

; Проверяем правую связь

mov bl, rl\_array + di;

cmp bl, al;

; Если не совпадение, то переходим к сравнению со следующим узлом

; Если совпадение, то добавляем единичку

jne right\_next;

; Теперь нашим текущим индексом становится тот узел, в котором мы обнаружили связь

mov al, i\_array + di;

; Запомним значение DI и занесём в него значение счётчика

push di;

push ax;

push dx;

mov ax, si;

mov bl, 8;

mul bl;

add ax, dx;

; AX = SI \* 8 + DX

mov di, ax;

mov code\_array + di, 31h;

inc di;

pop dx;

pop ax;

pop di;

; Смещаемся на одну ячейку

inc dx;

right\_next:

; До конца таблицы

inc di;

cmp di, cx;

jne codingInner;

continue:

; До конца таблицы

inc si;

cmp si, cx;

jne coding;

; Здесь же разворачиваем битовую последовательность в другую сторону

mov si, 0;

reverse:

xor ah, ah;

push si;

mov ax, si;

mov bl, 8;

mul bl;

mov di, ax;

add al, 8;

mov si, ax;

mov dl, 4;

turn:

dec si;

; Нужно учитывать факт того, что в наших восьми ячейках содержится и пустой код

mov al, code\_array + si;

; Достаточно проверить только AL, поскольку в нём - символы с конца

cmp al, 0;

je null\_char;

mov bl, code\_array + di;

mov code\_array + di, al;

mov code\_array + si, bl;

inc di;

null\_char:

dec dl;

cmp si, di;

jg turn;

pop si;

inc si;

cmp si, cx;

jne reverse;

pop di;

pop si;

pop dx;

pop cx;

pop bx;

pop ax;

ENDM

writefile\_cp MACRO handle, c\_array, code\_array, len, enter\_

local writing, writing\_inner, continue, null\_char;

push ax;

push bx;

push cx;

push dx;

push si;

push di;

xor ax, ax;

mov ax, len;

mov di, ax;

mov si, 0

writing:

; Признак конца таблицы

cmp c\_array + si, 0;

je continue;

; Пишем символ

mov ah, 40h;

mov bx, handle;

mov cx, 1;

lea dx, c\_array + si;

int 21h;

push di;

push si;

mov ax, si;

mov bl, 8;

mul bl;

mov si, ax;

mov di, 0;

; Пишем коды

writing\_inner:

mov ah, 40h;

mov bx, handle;

mov cx, 1;

lea dx, code\_array + si;

int 21h;

inc si;

inc di;

cmp di, 8;

jne writing\_inner;

pop si;

pop di;

; Пишем перенос строки

mov ah, 40h;

mov bx, handle;

mov cx, 2;

lea dx, enter\_;

int 21h;

continue:

inc si;

cmp si, di;

jne writing;

pop di;

pop si;

pop dx;

pop cx;

pop bx;

pop ax;

ENDM

writefile\_encode MACRO handle, string, c\_array, code\_array, enter\_, len

local encoding, searching, continue, not\_found, null\_char, coding\_end, encoding\_inner, ending;

push ax;

push bx;

push si;

push di;

mov si, 0;

encoding:

mov ah, string + si;

; Признак конца строки

cmp ah, 0;

je ending;

; Запоминаем SI

push si;

; Ищем символ в кодовой таблице

mov si, 0;

searching:

mov al, c\_array + si;

cmp ah, al;

jne not\_found;

; AX с символами нам больше не нужен, поскольку мы нашли совпадение и можем вычислить все индексы

mov ax, si;

mov bl, 8;

mul bl;

mov si, ax;

mov di, 0;

; Записываем кодовую последовательность

encoding\_inner:

cmp code\_array + si, 0

je continue;

mov ah, 40h;

mov bx, handle;

mov cx, 1;

lea dx, code\_array + si;

int 21h;

inc di;

inc si;

cmp di, 8;

je continue;

jmp encoding\_inner;

not\_found:

inc si;

cmp si, len;

je continue;

jmp searching;

continue:

pop si;

inc si;

jmp encoding;

ending:

pop di;

pop si;

pop bx;

pop ax;

ENDM

getcodes MACRO buffer, c\_array, code\_array, len

local codes, coding, next, end\_of\_code, ending, continue;

push ax;

push cx;

push si;

push di;

local char\_length: word;

mov char\_length, 0;

mov si, 0;

mov di, 0;

codes:

mov ah, buffer + si;

; Признак конца буфера

cmp ah, 0;

je ending;

inc char\_length;

mov c\_array + di, ah;

; Запоминаем DI и начинаем обращение к массиву кодов

push di;

mov ax, di;

mov bl, 8;

mul bl;

mov di, ax;

mov cx, 0;

; Начинаем с символа, следующего за буквой (в файле они "слеплены")

; Заносим все 8

coding:

inc si;

mov ah, buffer + si;

mov code\_array + di, ah;

inc di;

inc cx;

cmp cx, 8;

jne coding;

pop di;

inc di;

; Проскакиваем связку 13-10

add si, 3;

jmp codes;

ending:

mov ax, char\_length;

mov len, ax;

pop di;

pop si;

pop ax;

ENDM

decodestring MACRO buffer, c\_array, code\_array, r\_buffer, handle, len

local decoding, ending, decoding\_inner, not\_equal, found, found\_continue, next\_char, next, self\_clear, continue, checking;

push ax;

push bx;

push si;

push di;

local code\_length: word;

mov ax, len;

mov bl, 8;

mul bl;

mov code\_length, ax;

mov si, 0;

mov di, 0;

decoding:

mov ah, buffer + si;

cmp ah, 0;

je ending;

; Будем записывать в буфер каждый новый символ и проверять на совпадение со всей считанной из файла таблицей

; Если найдём совпадение, то отправляем в файл символ, обнуляем буфер и продолжаем

mov r\_buffer + di, ah;

; Сверяем содержимое восьмибайтного буфера со всей кодовой таблицей

; В DI будет кол-во символов в буфере, в SI - текущее положение

push si;

push di;

mov si, 0;

mov di, 0;

; Проходим по всей кодовой таблице

decoding\_inner:

push si;

add si, di;

mov ah, code\_array + si;

pop si;

cmp ah, r\_buffer + di;

; Если несовпадение, то переходим к следующему символу

jne next\_char;

; Если совпадение, то продолжаем сравнивать

inc di;

cmp di, 8;

; Если мы прошли восемь ячеек и таким образом получили полное совпадение

je found;

; Иначе просто продолжаем

jmp next;

found:

; Надо поделить SI на 8 из-за того, что мы индексировали с его помощью кодовый массив

mov ax, si;

mov bl, 8;

div bl;

mov si, ax;

mov ah, 40h;

mov bx, handle;

mov cx, 1;

lea dx, c\_array + si;

int 21h;

; Теперь надо очистить буфер

mov si, 0;

self\_clear:

mov r\_buffer + si, 0;

inc si;

cmp si, 8;

jne self\_clear;

pop di;

mov di, 0;

; Если мы нашли совпадение по символу, то нам требуется не inc di, а обнуление,

; поскольку мы уже очистили буфер и нужно начинать запись заново

jmp found\_continue;

next\_char:

; Если мы получили несовпадение, то надо обнулить DI, а SI добавить 8 (перейти к следующему символу)

mov di, 0;

add si, 8;

next:

; Сравниваем с концом кодового массива

cmp si, code\_length;

jge continue;

jmp decoding\_inner;

continue:

pop di;

inc di;

found\_continue:

pop si;

inc si;

jmp decoding;

ending:

pop di;

pop si;

pop bx;

pop ax;

ENDM

# **Файл print\_m.asm**

В данный файл вынесены все макросы вывода на консоль, которые использовались в процессе написания работы для проверки корректности процедур:

; Вывести символ

printc MACRO C

push ax;

push dx;

xor ax, ax;

xor dx, dx;

mov ah, 02h;

mov dl, C;

int 21h;

pop dx;

pop ax;

ENDM

; Вывести строку

prints MACRO S

local printString, endOfString;

push ax;

push si;

xor ax, ax;

xor si, si;

local char: byte; локальная переменная в рамках макроса

mov si, 0;

printString:

mov ah, S + si;

cmp ah, 0; признак конца строки - код символа знака доллара

je endOfString; если конец строки, то цикл из безусловных прыжков прерывается

mov char, ah;

printc char; пользуемся собственным макросом

inc si;

jmp printString;

endOfString:

printc 13;

printc 10;

pop si;

pop ax;

ENDM

printa MACRO array, len

local printing, ending;

push ax;

push dx;

push si;

mov si, 0;

mov ah, 02h; код функции

printing:

mov dl, array + si; текущая ячейка

int 21h;

printc 20h "разлепим" ячейки с помощью пробелов

inc si;

mov dx, len;

cmp si, dx; признак конца массива (в основном файле заполнен нулями)

je ending; прыжок за границы бесконечного цикла

jmp printing;

ending:

printc 13;

printc 10;

pop si;

pop dx;

pop ax;

ENDM

printindexes MACRO i\_array, len

local printing, ending;

push ax;

push si;

push di;

local num: byte;

mov ax, len;

mov di, ax;

mov si, 0;

printing:

mov al, i\_array + si;

mov num, al;

cmp si, di;

je ending;

printn num; отправляем индекс на вывод

printc 20h;

inc si;

jmp printing;

ending:

printc 13;

printc 10;

pop di;

pop si;

pop ax;

ENDM

printn MACRO N

local transferDigitsToStack, printDigits;

push ax;

push bx;

push cx;

push dx;

push si;

xor ax, ax;

xor bx, bx;

xor cx, cx;

xor dx, dx;

xor si, si;

local char: byte;

; Процедура печати многозначного числа на консоль похожа на таковую в языках высокого уровня:

; мы делим число на 10 до тех пор, пока не получим ноль, а каждый остаток отправляем в стек;

; из-за того, что мы будем вытягивать цифры из стека задом наперёд - а в стек они приходили с конца, -

; мы получим необходимое нам число.

mov si, 0;

mov al, N;

transferDigitsToStack:

xor dx, dx;

inc si;

mov bx, 10;

div bx;

mov bx, dx;

push bx;

cmp al, 0;

jne transferDigitsToStack;

mov cx, si;

printDigits:

pop dx;

add dl, 030h;

mov char, dl;

printc char;

loop printDigits;

pop si;

pop dx;

pop cx;

pop bx;

pop ax;

ENDM

printtree MACRO len, c\_array, f\_array, i\_array, ll\_array, rl\_array

local printing, ending, not\_space\_i, not\_space\_f, not\_space\_l;

push ax;

push si;

push di;

local argument: byte;

mov al, len;

xor ah, ah;

mov di, ax;

mov si, 0;

printing:

printc 20h;

mov al, i\_array + si;

mov argument, al;

printn argument;

; Для форматирования вывода добавим дополнительный пробел после однозначных чисел

cmp al, 9;

jg not\_space\_i;

printc 20h;

not\_space\_i:

printc 20h;

mov al, c\_array + si;

mov argument, al;

printc argument;

printc 20h;

mov al, f\_array + si;

mov argument, al;

printn argument;

; Для форматирования вывода добавим дополнительный пробел после однозначных чисел

cmp al, 9;

jg not\_space\_f;

printc 20h;

not\_space\_f:

printc 20h;

mov al, ll\_array + si;

mov argument, al;

printn argument;

; Для форматирования вывода добавим дополнительный пробел после однозначных чисел

cmp al, 9;

jg not\_space\_l;

printc 20h;

not\_space\_l:

printc 20h;

mov al, rl\_array + si;

mov argument, al;

printn argument;

printc 13;

printc 10;

inc si;

cmp si, di;

je ending;

jmp printing;

ending:

pop di;

pop si;

pop ax;

ENDM

printcodes MACRO c\_array, code\_array, len

local printing, codes, continue, ending;

push ax;

push bx;

push si;

push di;

local argument: byte;

mov bx, len;

mov si, 0;

printing:

mov al, c\_array + si;

cmp al, 0;

je continue;

mov argument, al;

printc argument;

printc 20h;

mov di, 0;

codes:

push di;

push bx;

mov ax, si;

mov bl, 8;

mul bl;

add ax, di;

mov di, ax;

mov al, code\_array + di;

mov argument, al;

printc argument;

pop bx;

pop di;

inc di;

cmp di, 8;

jne codes;

printc 13;

printc 10;

continue:

inc si;

cmp si, bx;

je ending;

jmp printing;

ending:

pop di;

pop si;

pop ax;

ENDM

# **Файл coding.asm**

Файл для кодирования по алгоритму Хаффмана.

INCLUDE \..\Programs\macro.asm

INCLUDE \..\Programs\print\_m.asm

.MODEL SMALL

.STACK 100h

.DATA

buffer DB 1024 dup (0)

fileName DB 'Huffman.txt', 0

fileHandle DW ?

newFileName DB 'Encoding.txt', 0

newFileHandle DW ?

cpFileName DB 'cp.txt', 0

cpFileHandle DW ?

\_length DW 1024

len DW 10

symbolsArray DB 120 dup (0)

freqArray DB 120 dup (0)

indexArray DB 120 dup (0)

leftLinkArray DB 120 dup (0)

rightLinkArray DB 120 dup (0)

codeArray DB 960 dup (0)

enter\_ DB 13, 10

.CODE

START:

; Инициализация сегмента данных и базовая очистка регистров перед началом работы

mov ax, @Data

mov ds, ax

xor ax, ax

xor bx, bx

xor cx, cx

xor dx, dx

xor si, si

xor di, di

openfile fileName

mov fileHandle, ax; после работы макроса в регистре AX находится дескриптор файла\

readfile fileHandle, \_length, buffer

freq\_co buffer, symbolsArray, freqArray, len

sorta freqArray, symbolsArray, indexArray, len

tree len, symbolsArray, freqArray, indexArray, leftLinkArray, rightLinkArray

huffcodes len, symbolsArray, codeArray, indexArray, leftLinkArray, rightLinkArray

createfile cpFileName

mov cpFileHandle, ax;

createfile newFileName

mov newFileHandle, ax;

writefile\_cp cpFileHandle, symbolsArray, codeArray, len, enter\_

writefile\_encode newFileHandle, buffer, symbolsArray, codeArray, enter\_, len

; Выход из программы

mov ax, 4c00h

int 21h

END START;

# **Файл decoding.asm**

Файл возвращения закодированной строки в изначальный вид.

INCLUDE \..\Programs\macro.asm

INCLUDE \..\Programs\print\_m.asm

.MODEL SMALL

.STACK 100h

.DATA

buffer DB 1024 dup (0)

destFileName DB 'Decoding.txt', 0

destFileHandle DW ?

sourceFileName DB 'Encoding.txt', 0

sourceFileHandle DW ?

cpFileName DB 'cp.txt', 0

cpFileHandle DW ?

\_length DW 1024

len DW 10

symbolsArray DB 120 dup (0)

codeArray DB 960 dup (0)

enter\_ DB 13, 10

readBuffer DB 8 dup (0)

.CODE

START:

; Инициализация сегмента данных и базовая очистка регистров перед началом работы

mov ax, @Data

mov ds, ax

xor ax, ax

xor bx, bx

xor cx, cx

xor dx, dx

xor si, si

xor di, di

openfile cpFileName

mov cpFileHandle, ax;

openfile sourceFileName

mov sourceFileHandle, ax;

createfile destFileName

mov destFileHandle, ax;

readfile cpFileHandle, \_length, buffer

getcodes buffer, symbolsArray, codeArray, len

readfile sourceFileHandle, \_length, buffer

decodestring buffer, symbolsArray, codeArray, readBuffer, destFileHandle, len

; Выход из программы

mov ax, 4c00h

int 21h

END START;